



# gRPC

Benyamin Mousavi



1. آشنایی با RPC
2. آشنایی با مفاهیم gRPC
3. آشنایی با HTTP2
4. مزایا و معایب gRPC
5. موارد استفاده

1. آشنایی با ویژگی ها و عملکرد gRPC حیاتی
2. دلیل حیاتی بودن تفاوت با تکنولوژی های مورد استفاده
3. gRPC وابستگی به زبان ندارد

# معرفی RPC

1. مخفف Remote Procedure Call

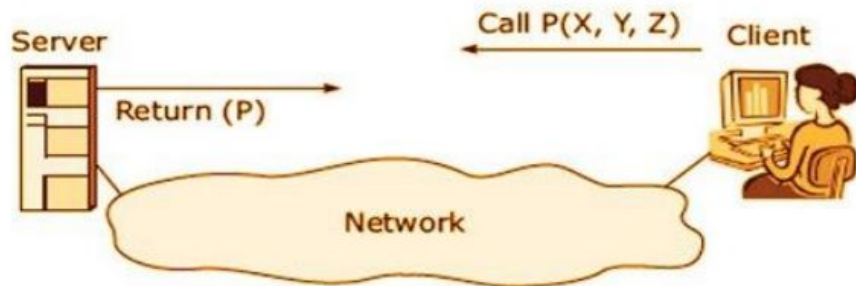
2. صدا زدن توابع با پرتوکل مبتنی بر شبکه

3. روش توسعه Server/Client

4. معرفی در 1976

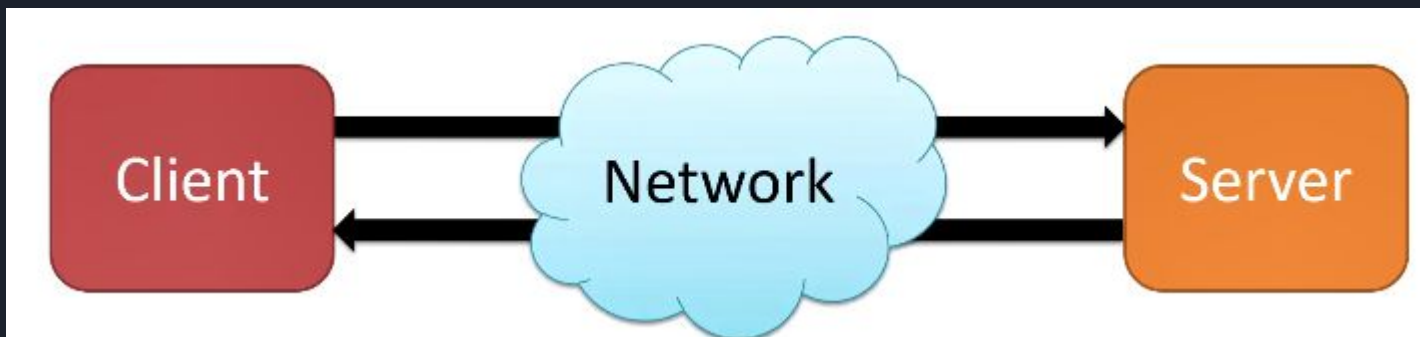
5. RFC 707

## Remote Procedure Call (RPC)



# روش کار

1. شروع کار با Client
2. صدا زدن تابعی در سرور
3. پردازش درخواست در سرور
4. توقف Client برای پاسخ
5. بازگشت نتیجه مناسب
6. بازگشت نتیجه مناسب





# تفاوت با Rest

1. متدی از یک موجودیت صدا زده می شود

2. در Rest هر موجودیت یک Resource (یعنی هر Resource یک URL دارد که از طریق Url ها به Resource می رسیم و از افعال HTTP استفاده میکنیم برای این که بگیم با اون Resource میخوایم چیکار کنیم اما در rpc اینطور نیست ما یک کلاس یا یک انتیتی داریم که متدهایی دارند و این متد ها را صدا میزنیم و این که پشت صحنه خیلی درگیر این که این متدها چطوری صدا زده میشوند روی شبکه نمیشویم )

3. استفاده از افعال HTTP



نکته:

با اینکه تفاوت دیدگاه و عملکرد وجود دارد اما هم Rest و هم gRPC از HTTP استفاده می کنند.

(در Rest مستقیم از این افعال استفاده میکنیم)



## نگاهی به تاریخچه gRPC

1. تاریخچه و مفاهیم کمک به درک بهتر
2. یک فریم ورک، Source Open
3. توسعه توسط گوگل در 2000
4. انتشار سورس در 2014
5. استفاده از Buffer Protocol برای IDL





6. انتقال پیام به صورت باینری توسط HTTP2
7. پشتیبانی از مجموعه وسیعی از زبان ها
8. تعریف چندین Procedure قابل صدا زدن
9. ارائه پیاده سازی توابع در سرور
10. قابلیت توسعه Client و سرور با زبان های مختلف

# آشنایی با Protocol Buffer

1. تعریف API ها توسط Buffer Protocol

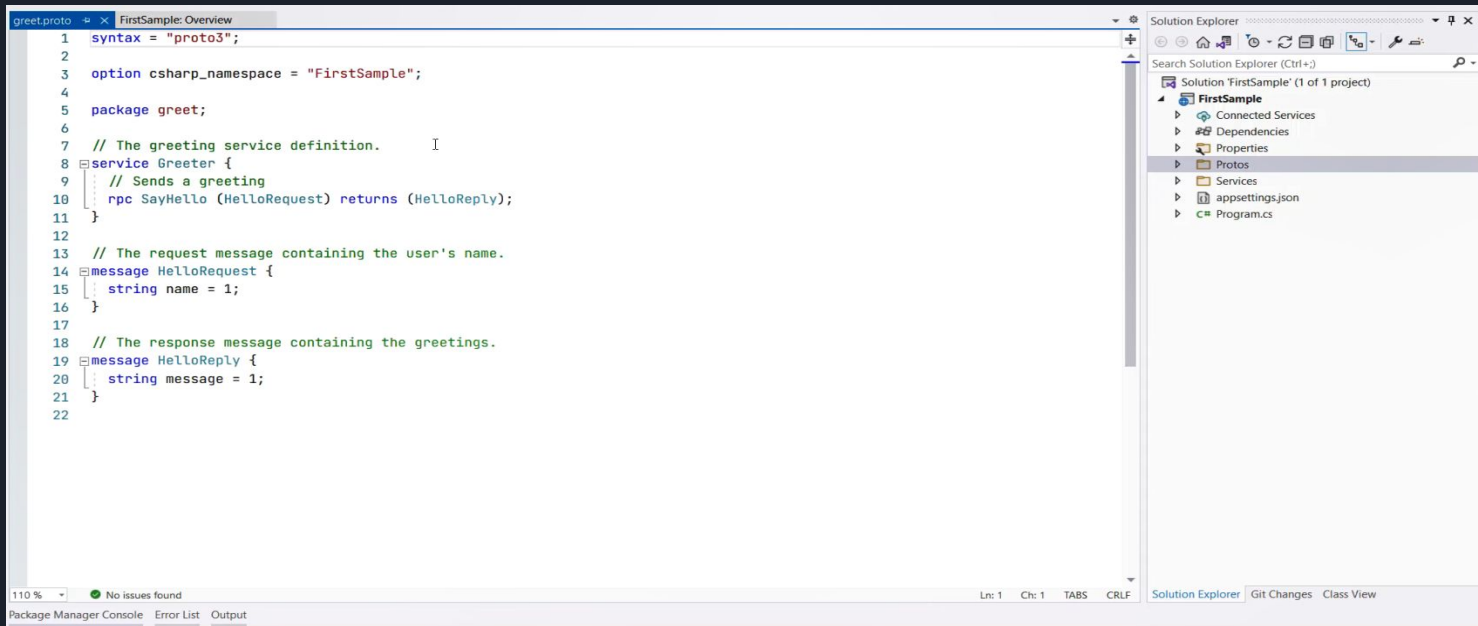
2. شناخت ویژگی های سرور توسط Client

3. یک زبان توصیف کننده

4. مانند WSDL برای SOAP

5. نگهداری با پسوند .proto






قسمت اول Syntax

قسمت دوم سرویس ها

قسمت سوم ساختار message ها



```
public class GreeterService : Greeter.GreeterBase
{
    private readonly ILogger<GreeterService> _logger;

    public GreeterService(ILogger<GreeterService> logger)
    {
        _logger = logger;
    }

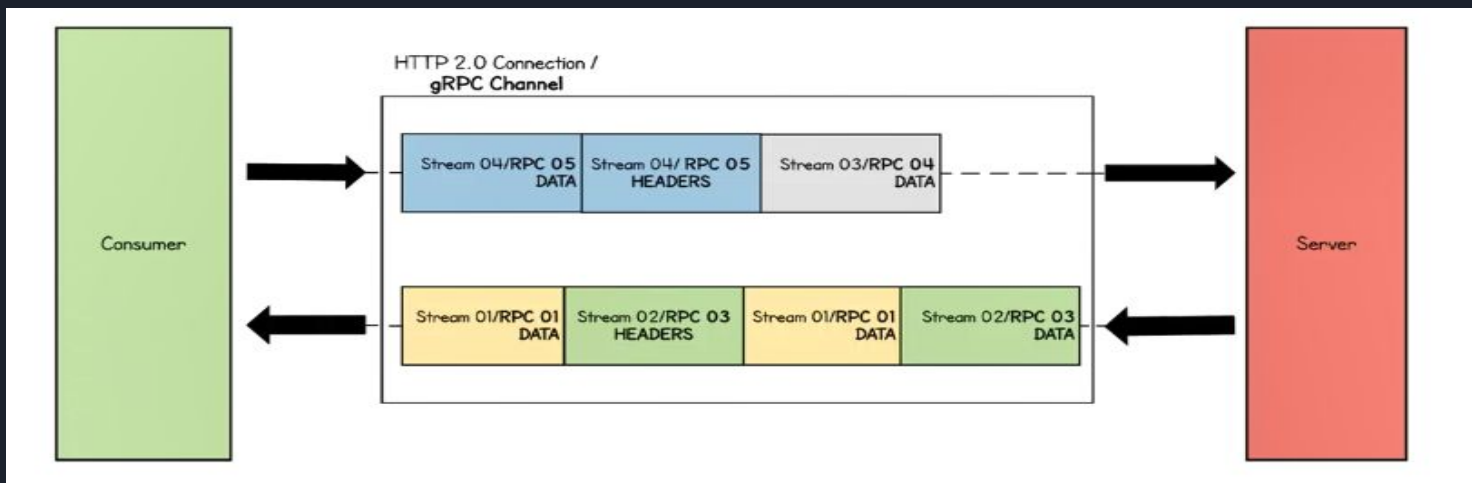
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)
    {
        return Task.FromResult(new HelloReply
        {
            Message = "Hello " + request.Name
        });
    }
}
```

پیاده سازی message ها

# آشنایی با Channel

1. ارسال درخواست از طریق Channel

2. ایجاد یک Connection به سرور





## اطلاعات مورد نیاز Channel

1. آدرس میزبان

2. پورت

3. Connection Credential

4. امکان تنظیم Credential برای Connection

5. امکان تنظیم Credential برای هر درخواست



## انواع قابل پشتیبانی (تنظیماتی که روی Channel میتونیم داشته باشیم)

1. SSL / TLS
2. ALTS یا Application Layer Transport security
3. Token based authentication

به ازای هر درخواست توکن را ارسال کنیم

کلاینتی که برای gRPC توسط ماکروسافت پیاده سازی میشه این قابلیت رو میده که خیلی راحت توکن را ست کنیم و از این به بعد این توکن روی هر درخواستمون رفت و آمد داشته باشد



## نکات تکمیلی

1. اغلب تنظیم روی هر درخواست
2. پیاده سازی توکار توسط مایکروسافت
3. تنظیمات دیگری مانند اندازه پیام و ... قابل انجام



# آشنایی با وضعیت های Channel

1. Idle : اتصال برقرار است اما در حالت استراحت است
2. Connecting : زمانی که کلاینت تلاش میکند به سرور متصل شود
3. Ready : میتونیم پیام بدیم و پیام بگیریم
4. TransentFailure : وقت هایی که خطا اتفاق افتاده

ولی Channel از بین نمیرود و مجدد به چرخه ی حیات خودش برگردد

5. Shutdown

1. ممکن است FataError اتفاق افتاده باشد

2. یا کار ما با Channel تمام شده باشد



## انواع سرویس ها

1. Unary
2. Server Streaming
3. Client Streaming
4. Bidirectional

# Unary



# Server Streaming

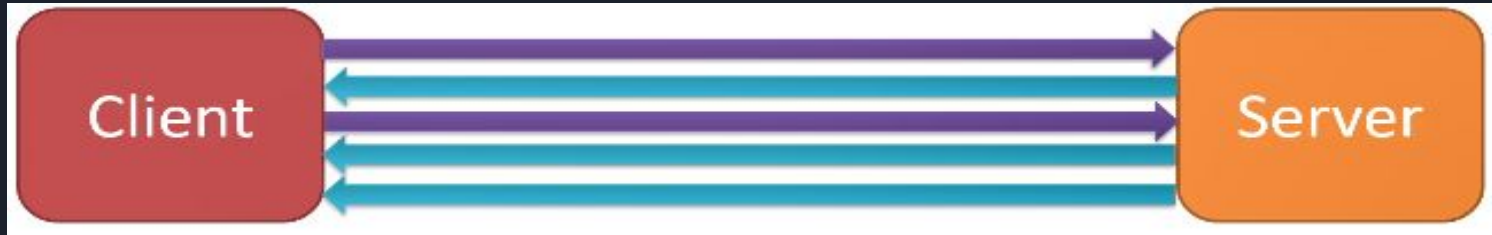


معمولا Status Code در آخرین پیام برمیگردد

# Client Streaming



# Bidirectional Streaming



## آشنایی با وضعیت های gRPC

Ok	DEADLINE_EXCEEDED	RESOURCE_EXHAUSTED	UNIMPLEMENTED
CANCEL	NOT_FOUND	FAILED_PRECONDITION	INTERNAL
UNKNOWN	ALREADY_EXISTS	ABORTED	UNAVAILABLE
INVALID_ARGUMENT	PERMISSION_DENIED	OUT_OF_RANGE	DATA_LOSS
			UNAUTHENTICATED



## نحوه کارکرد

1. استفاده از Header های HTTP
2. استفاده از Header ویژه Trailers
3. وضعیت درخواست در Trailers
4. پاسخ با فرمت Binary





## Trailers :

یک سری اطلاعات غیر از اطلاعات مرسومی که میخواهیم ارسال کنیم

یا دریافت کنیم

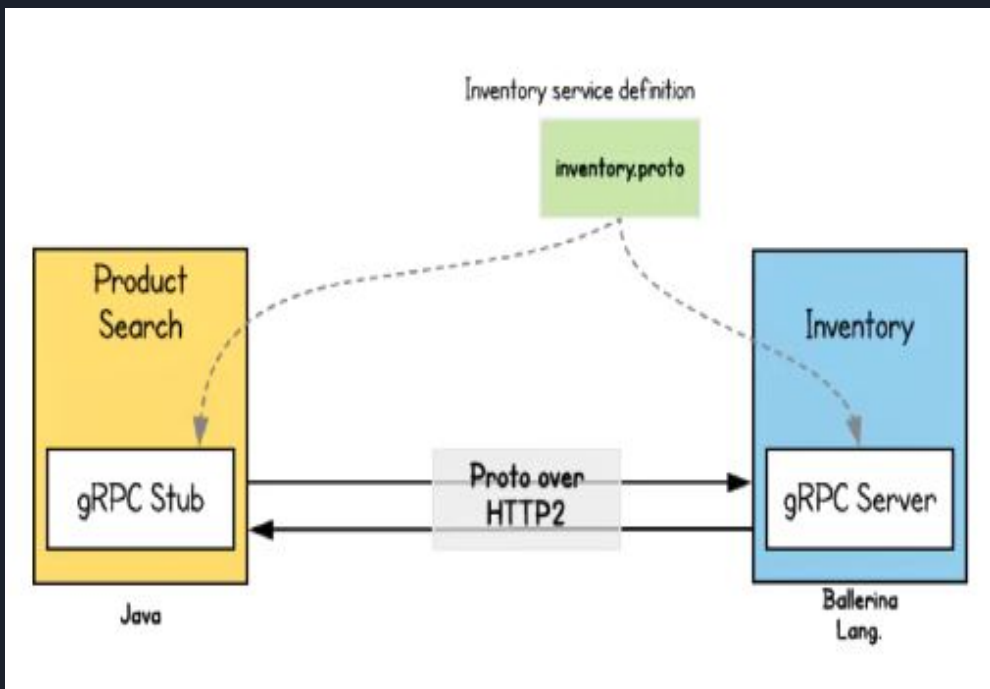
یعنی یا سرور ارسال میکند یا کلاینت ارسال میکند

این اطلاعات را با استفاده از استانداردهای HTTP از طریق Header جابجا میکند ولی در مورد Request و Response زمان جابجایی Trailer ها که یک سری Header های ویژه هستند تفاوت میکند یعنی وقتی Request ارسال میکند در ابتدا که Header های عادی را ارسال میکند Trailer ها را هم ارسال میکند

اما از سمت سرور که برمیگردد , Header های عادی اول برمیگردد بعد دیتا میرسد و در نهایت Trailer ها به دست ما میرسد .

یعنی زمانی ارسال میشود که مطمئن شویم که دیتا به درستی به دست کاربر رسیده باشد

## نحوه ارسال HTTP



1. امکان استفاده از TLS
2. ارسال درخواست POST
3. Application/grpc
4. application/grpc+proto
5. Application/grpc+json



# نحوه ارسال HTTP

6. استفاده از Authority Header

7. استفاده از Path برای شناسایی Proc

8. وضعیت پاسخ 200

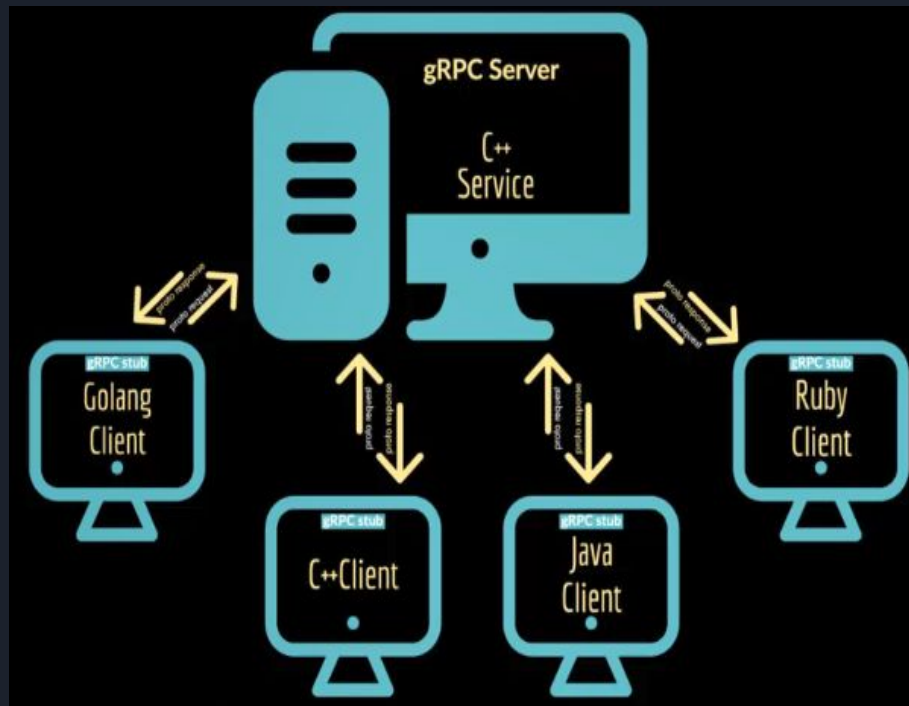
9. در صورت بروز خطا 503



## آشنایی با HTTP2

1. Multiplexing : چندین Request و Response با یک Connection ارسال و دریافت کنیم
2. Flow Control : الویت بندی درخواست ها و پاسخ ها
3. Compression and Binary data transport : توانایی ارسال Binary اطلاعات به صورت
4. Server Push : پوش کردن دیتا

## مزایا



1. حجم بهینه داده های ارسال
2. استریم دو طرفه
3. امنیت بهتر داده های Binary



## معایب

1. عدم پشتیبانی از مرورگر
2. عدم خوانایی برای انسان
3. غیر قابل Cache

1. ارتباط API با API
2. استفاده مناسب جهت Server Side APP
3. قابل استفاده با Background Job
4. استفاده پشت API Gateway